DEVIEW 2020

Bye Oracle, Hello PG:
쇼핑검색플랫폼, MSA로 새옷을 갈아입다

권동훈 / 오현진 쇼핑검색플랫폼

CONTENTS

DEVIEW 2020

Bye Oracle, Hello PG

네이버 쇼핑검색 플랫폼

- 13억 상품
- 40만 입점 쇼핑몰
- 1,800만/주 모바일 방문자수
- 5.2억/주 상품 클릭

DEVIEW 2020

Bye Oracle

- 지속적으로 상품 증가 추세(13억~)
- 지속적으로 늘어나는 하루 업데이트 이벤트(6억~)
- Scale-Up 한계
- Single point of failure

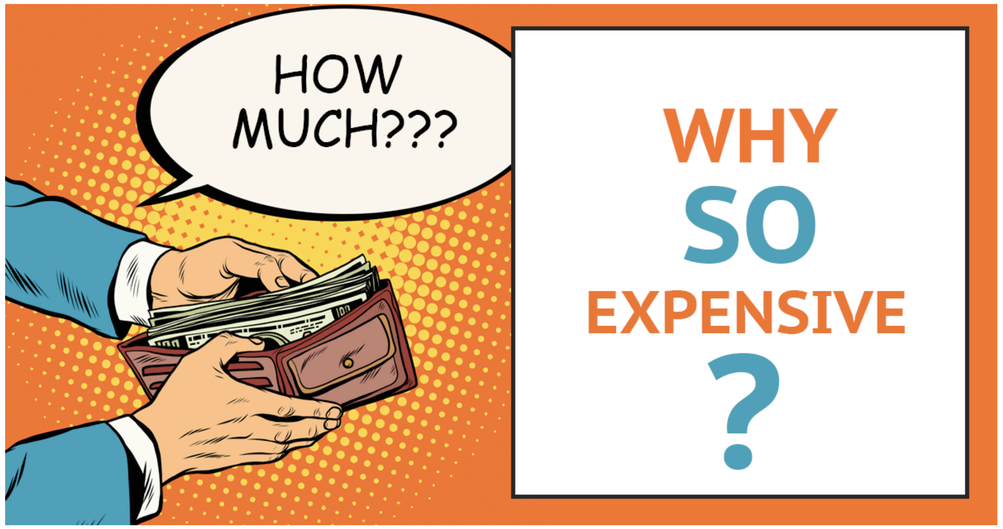
Bye Oracle

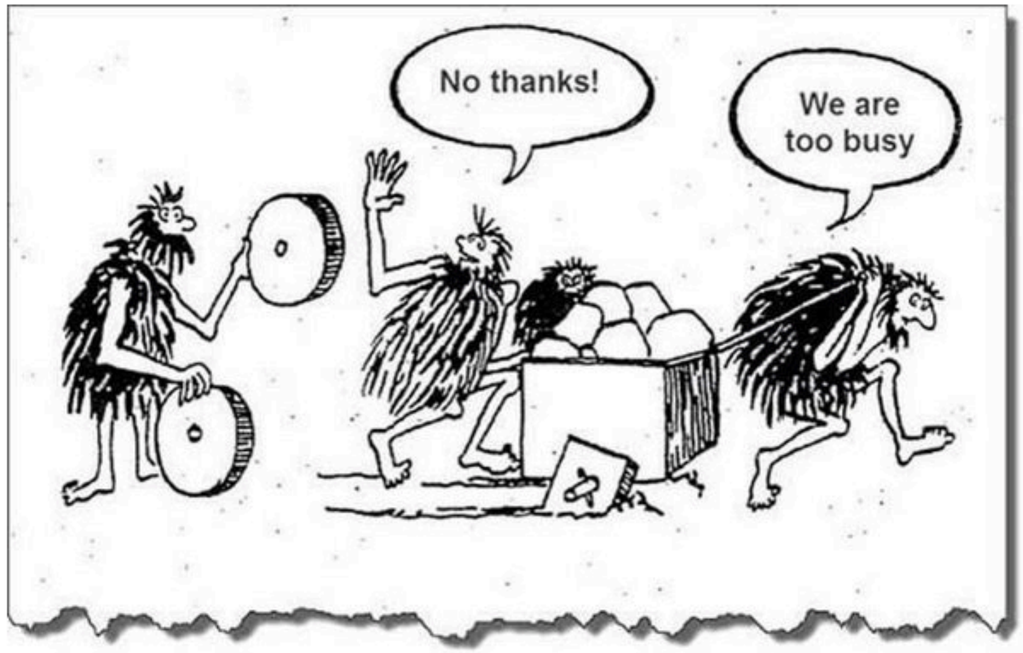
Oracle을 대체할 DB 선택 시, 가장 고려한 사항

- Distributed Database/Scale-out
- CDC(Change Data Capture)
- RDBMS

Oracle에서도 모두 지원하지 않나?

- CDC -> Golden-Gate
- Oracle Shading solution



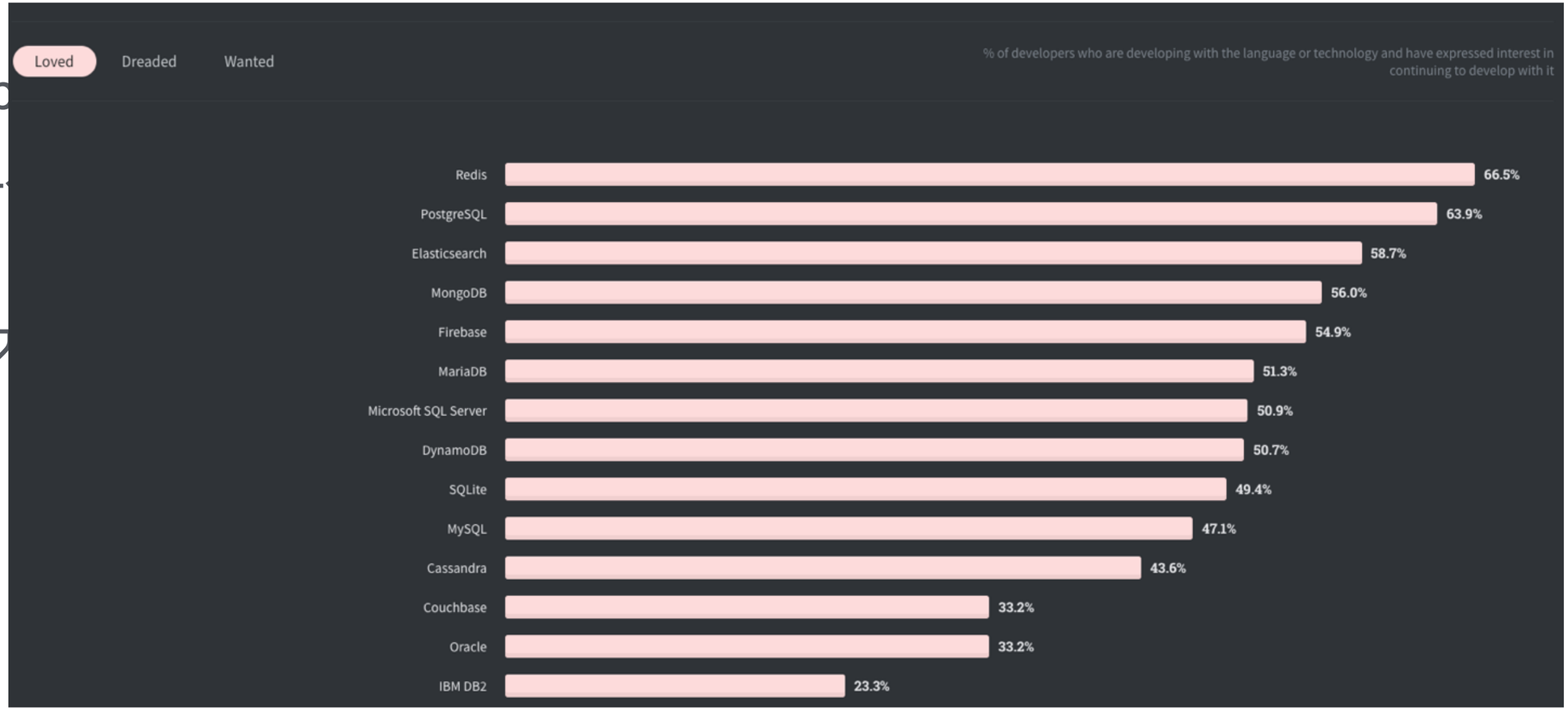


Hello PG

DEVIEW 2020

PostgresSQL을 선택한 이유는?

- Distributed Datab
- CDC 파이프라인 구
- 인기도 상승중
  - 개발자 커뮤니티



https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-databases

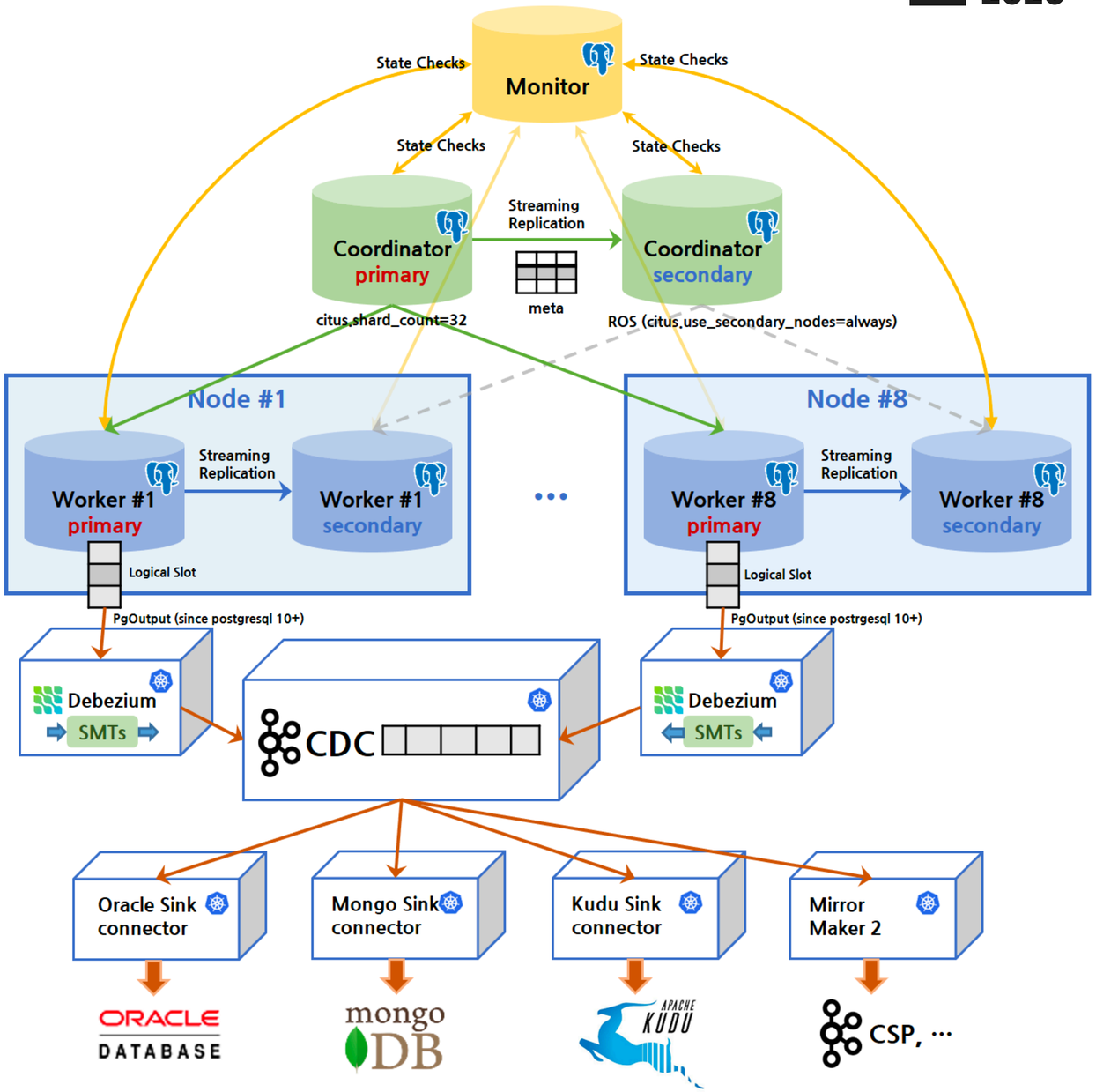
Bye Oracle, Hello PG

2019.08 ~ 2020.05

30여명의 개발자

500여개의 테이블 마이그레이션

3000여 개의 SQL 구문 변경



DEVIEW 2020

CITUS Distributed Database

CITUS

An RDBMS is a general-purpose data platform

But RDBMS's don't scale, right?

RDMBS's are hard to scale

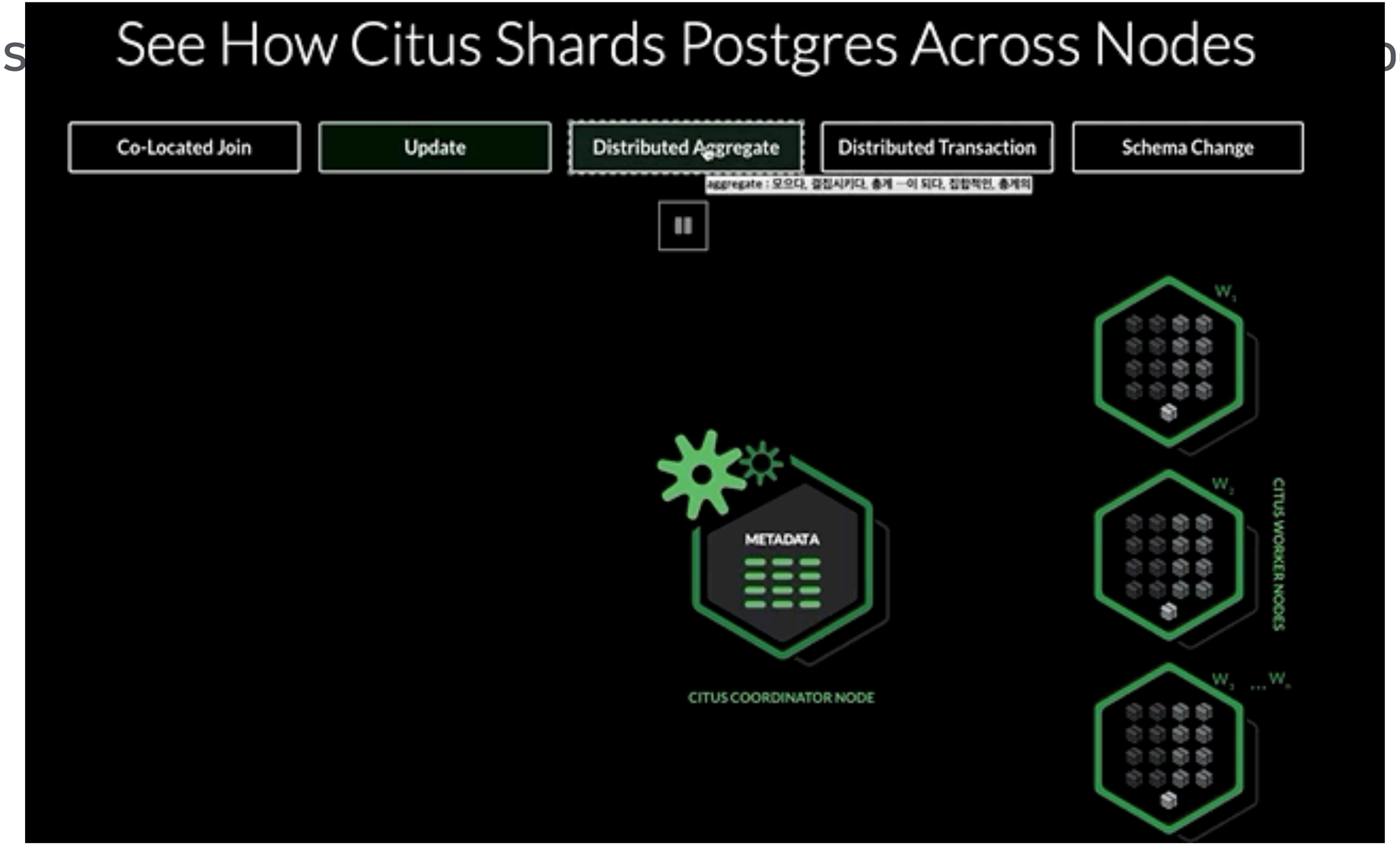
Distributed table

Optimize distributed sql

Distributed transaction

CITUS

Extens



https://www.citusdata.com/product#citus-diagram

Distributed Table

DEVIEW 2020

distributed table

- Distribution column
- hash partition
- hash token ranges assigned to the shard
- A unique index or primary key must contains the distribution column

```
create table product( product_id bigint primary key ...);
select create_distributed_table('product', 'product_id');
```

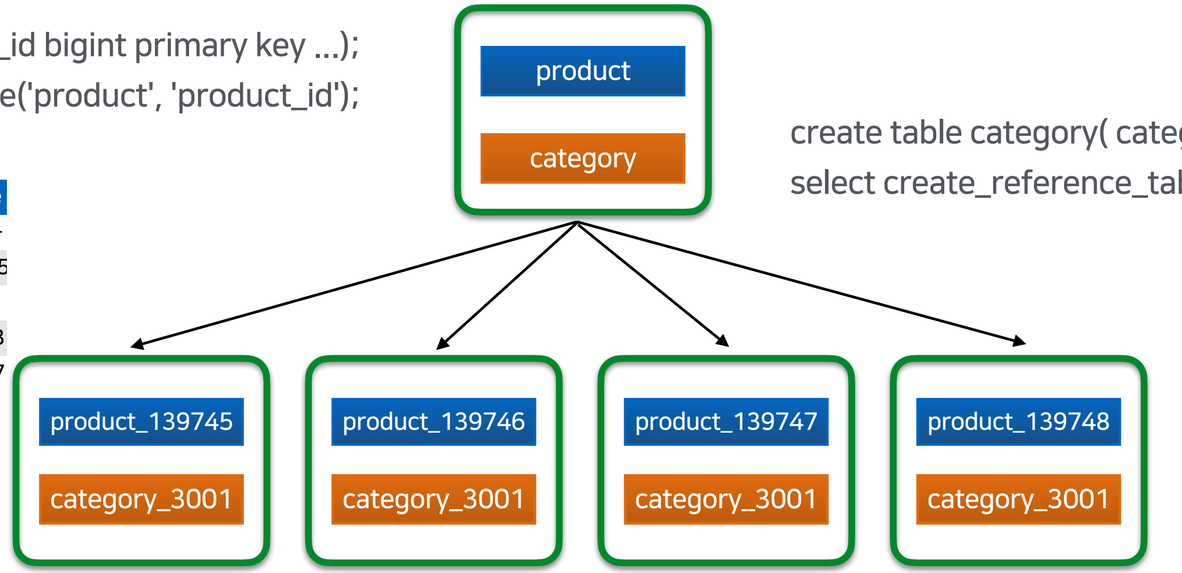
logicalrelid	shardid	shardstorage	shardminvalue	shardmaxvalue
product	139745	t	-2147483648	-1073741825
product	139746	t	-1073741824	-1
product	139747	t	0	1073741823
product	139748	t	1073741824	2147483647

logicalrelid	shardid	nodename
product	139745	worker004-shop
product	139746	worker006-shop
product	139747	worker008-shop
product	139748	worker010-shop

reference table

- Single shard
- Replicated on every worker
- statement-based replication(2pc)

```
create table category( category_id bigint primary key ...);
select create_reference_table('category');
```



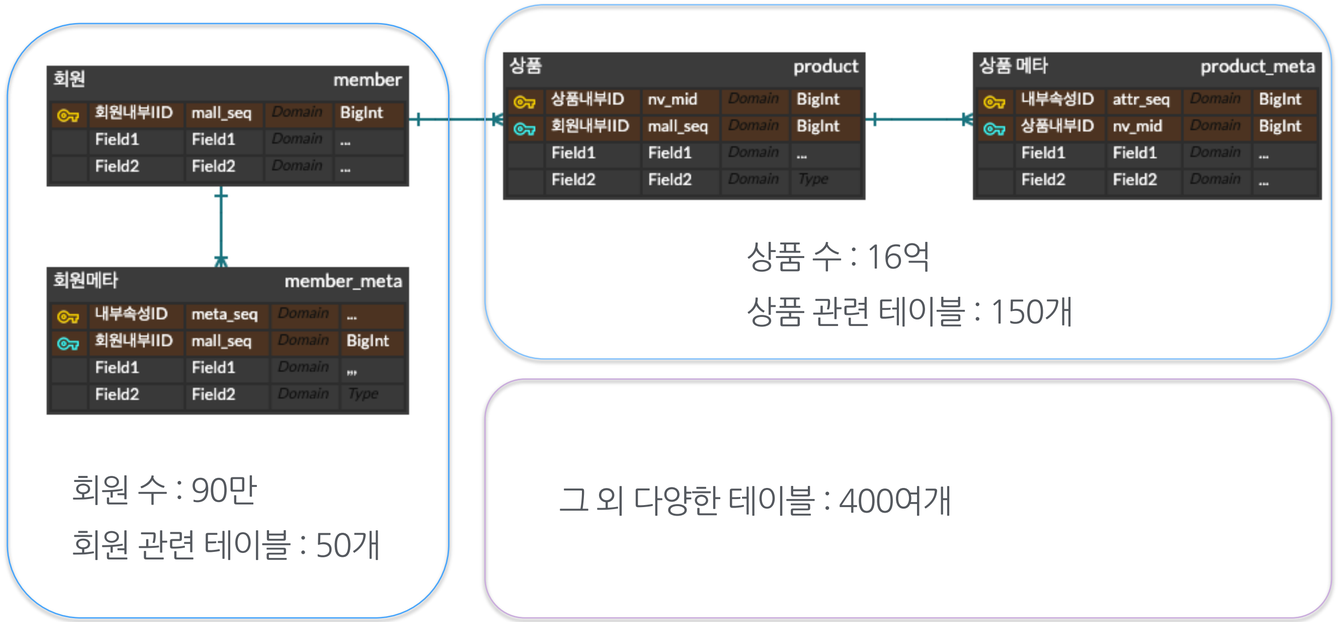
Distributed Table

DEVIEW 2020

Co-Location

Be sharded across machines in the same way

- Full query push down when filtering all tables by distributed column
- Efficient joins between distributed table on distributed column



- 1%의 회원이 전체 상품의 50%이상을 제공

Distributing Queries

DEVIEW 2020

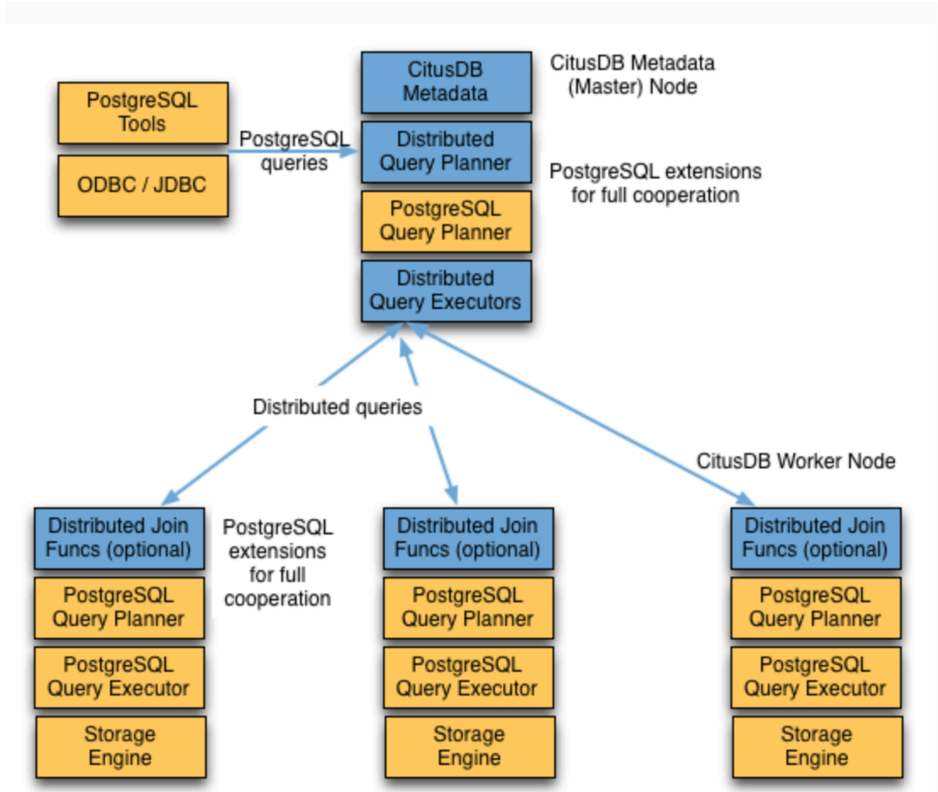
- Parallelism
  - Commutative property
  - associative property
  - distributed property
- Break the query into two parts
  - Coordinator query
  - Worker query fragments
- Optimization
  - Push down operations directly to related worker shards

$$Project_x(Collect(R_1, R_2, \dots)) = Collect(Project_x(R_1), Project_x(R_2) \dots)$$

$$SUM(x)(Collect(R_1, R_2, \dots)) = SUM(Collect(SUM(R_1), SUM(R_2) \dots))$$

$$Collect(R_1, R_2, \dots) \times Collect(S_1, S_2, \dots) = Collect(R_1 \times S_1, R_2 \times S_2, \dots)$$

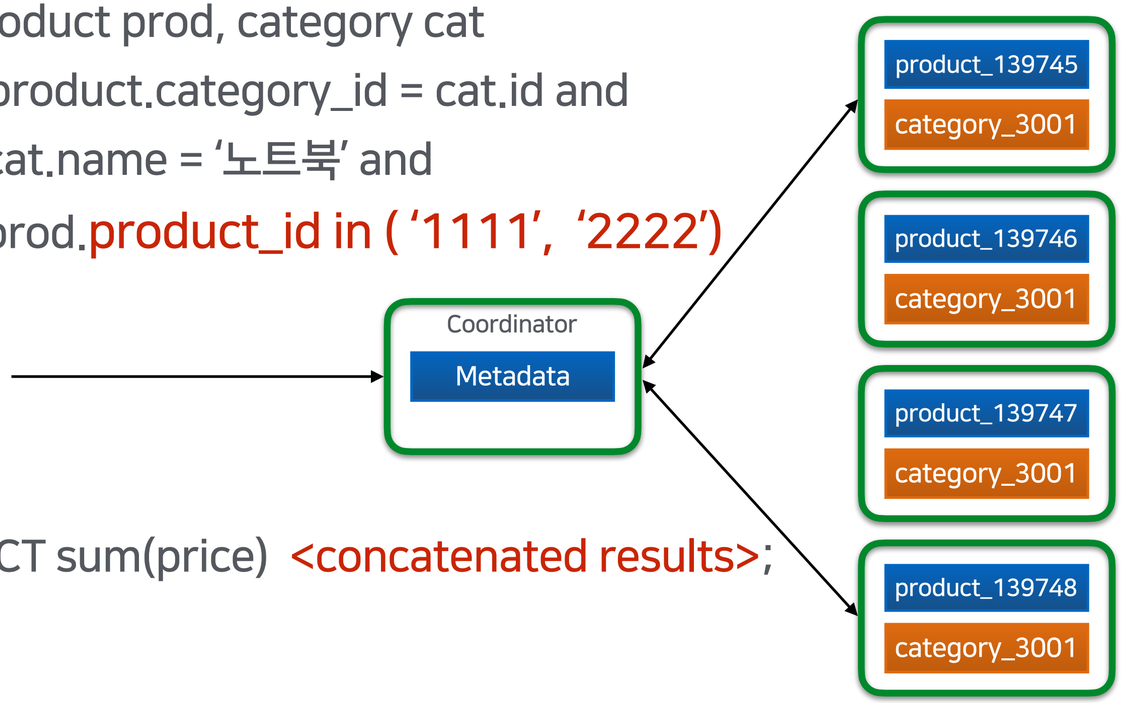
X = Join Operator



https://docs.citusdata.com/en/v9.4/develop/reference_processing.html?highlight=QUERY%20EXECUTE#query-processing

Distributing Queries


```
SELECT sum(product.price)
FROM product prod, category cat
WHERE product.category_id = cat.id and
      cat.name = '노트북' and
      prod.product_id in ('1111', '2222')
```



SELECT sum(price) <concatenated results>;

```
SELECT sum(product.price) as price
FROM product_139745 prod
JOIN category_2001 cat
ON (prod.category_id =cat.category_id)
WHERE   cat.name = '노트북' and
        prod.product_id in ('1111', '2222');
```

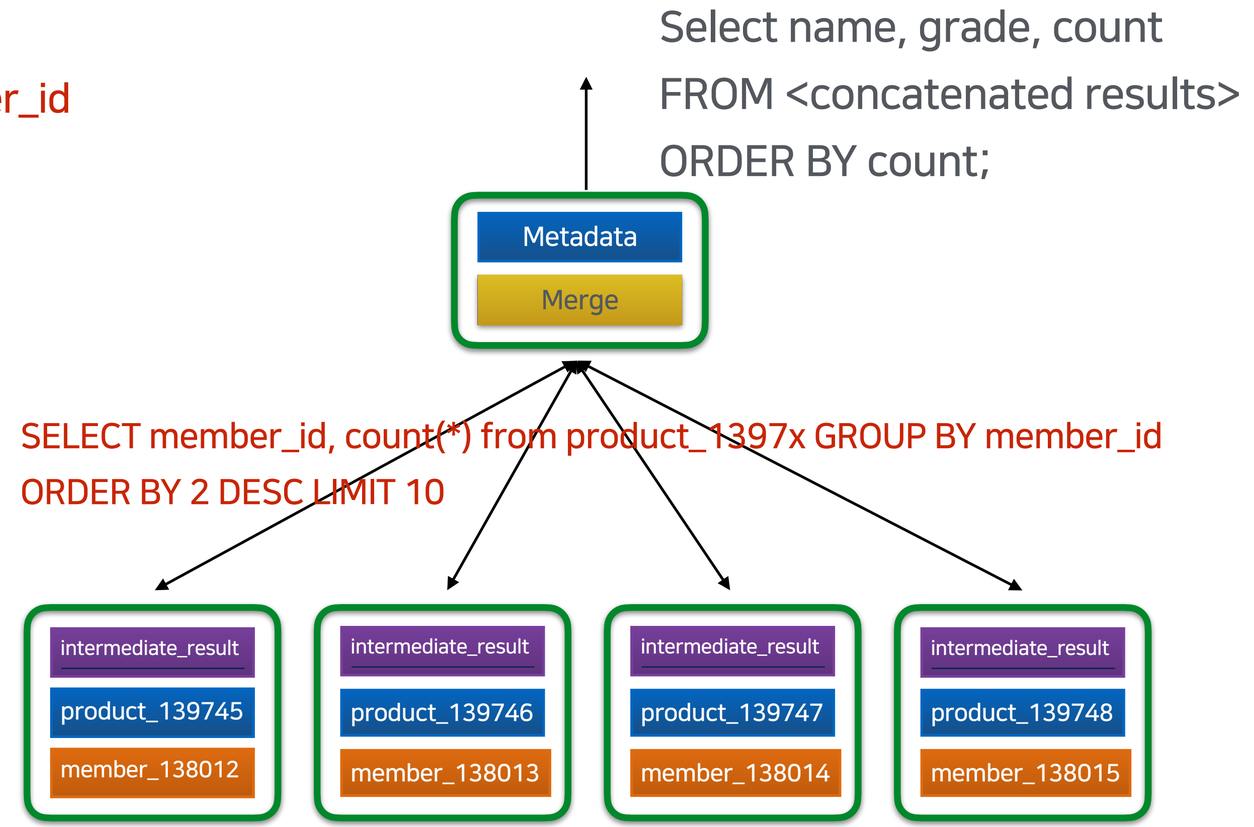
```
SELECT sum(product.price) as price
FROM product_139748 prod
JOIN category_2001 cat
ON (prod.category_id =cat.category_id)
WHERE   cat.name = '노트북' and
        prod.product_id in ('1111', '2222');
```

Distributing Queries

DEVIEW 2020

Push-Pull execution

```
SELECT member.name, member.grade, count from member
JOIN (
  SELECT member_id, count(*) from product GROUP BY member_id
  ORDER BY 2 DESC LIMIT 10
) top10_member
ON member.id = top10_member.member_id
ORDER BY count;
```

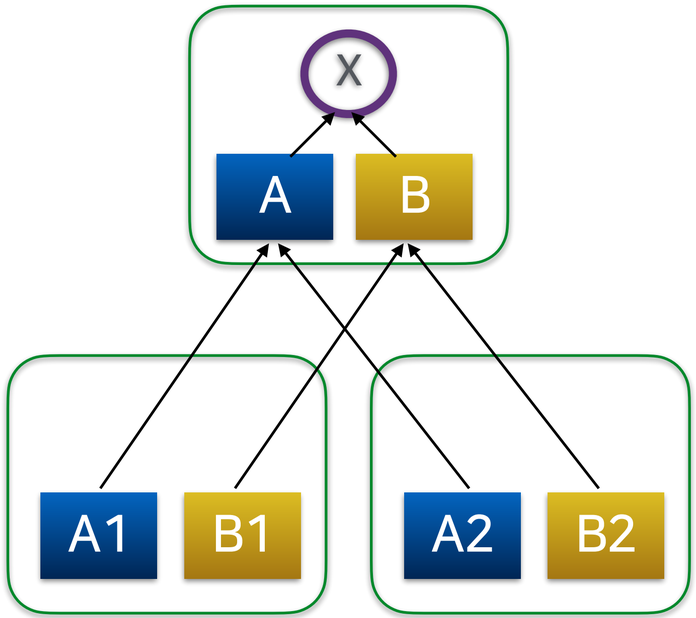


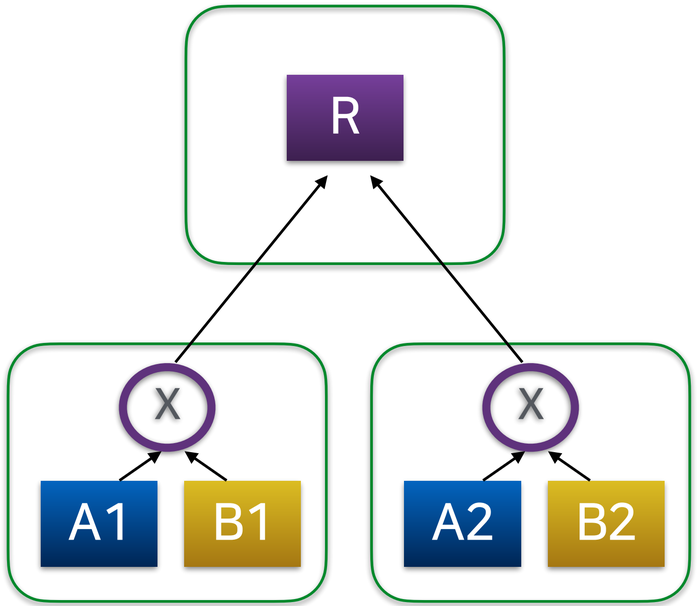

```
SELECT member.name, member.grade count
FROM member_138012
JOIN (
  select * from intermediate_result
) top10_member
ON member.id = top10_member.member_id;
```

Distributed Join

$$Collect(R_1, R_2, \dots) \times Collect(S_1, S_2, \dots) = Collect(R_1 \times S_1, R_2 \times S_2, \dots)$$

X = Join Operator





Distributed Join

DEVIEW 2020

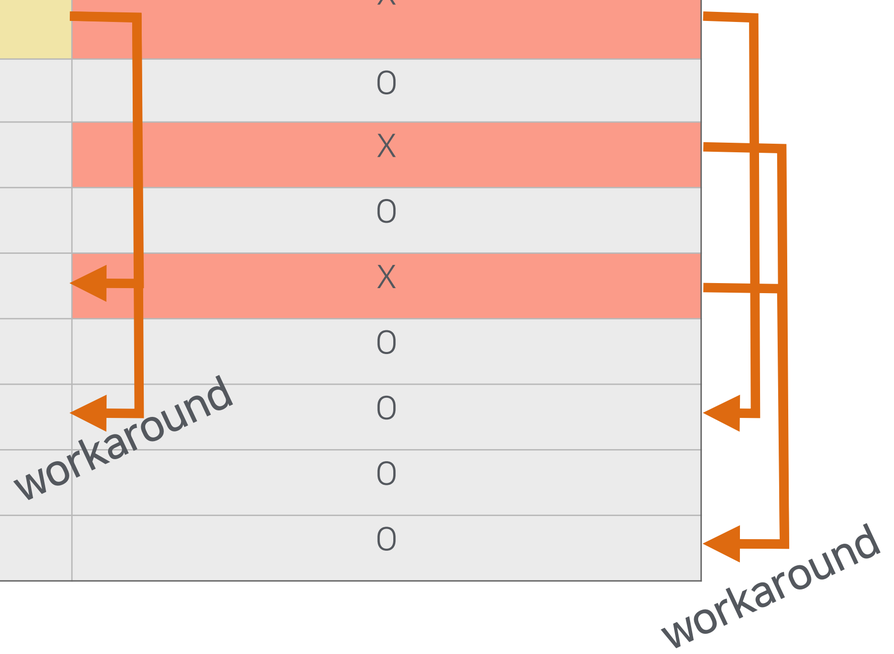
```
SELECT product.name, member.name, member.grade
FROM product
JOIN member
ON( product.member_id = member.id )
WHERE member.reg_ymdt > '2020-10-01'
ORDER BY product.price DESC
LIMIT 10;
```

```
SELECT product.name, member.name, member.grade
FROM product
JOIN (
    SELECT id, name, grade from member
    WHERE member.reg_ymdt > '2020-10-01'
) member
ON( product.member_id = member.id )
ORDER BY product.price DESC
LIMIT 10;
```

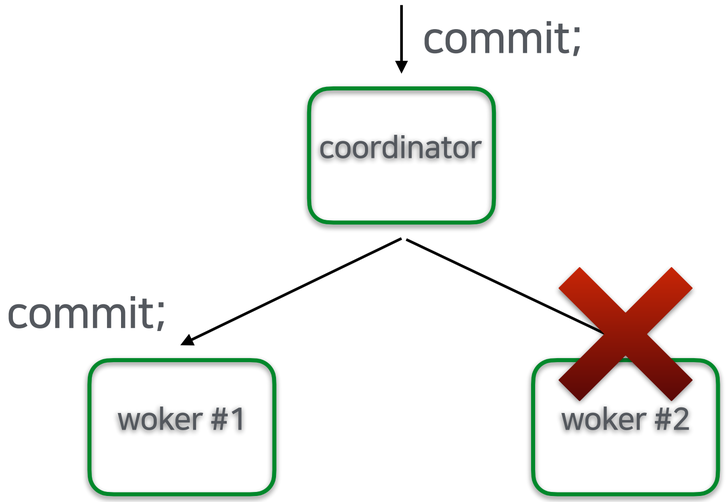
Distributed Join

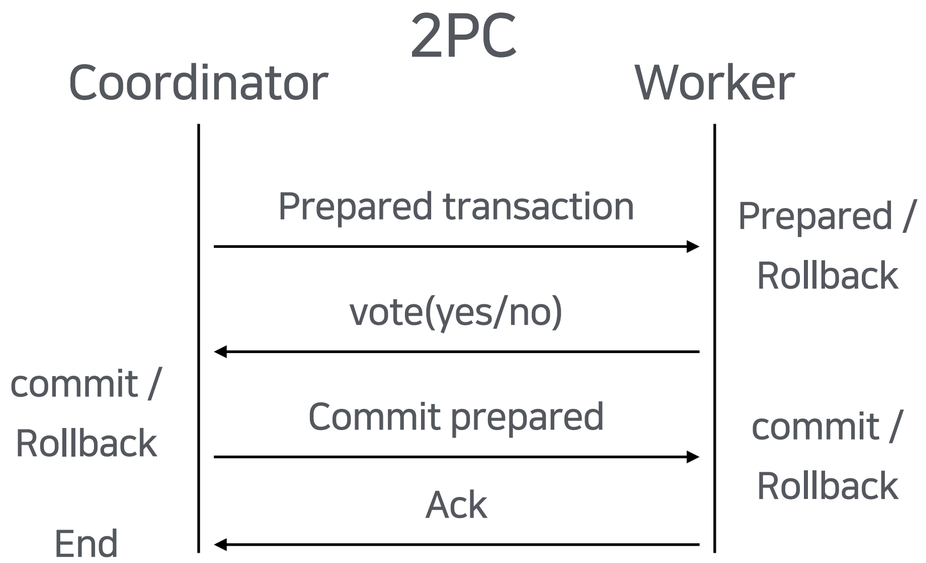
DEVIEW 2020

	INNER JOIN	OUTER JOIN
distributed table x distributed table (co-location)	O	O
distributed table x distributed table (non co-location)	X (O ㄴw repartition_joins)	X
distributed table x reference table	O	O
reference table x distributed table	O	X
reference table x reference table	O	O
Intermediate result x distributed table	O	X
Intermediate result x reference table	O	O
distributed table x Intermediate result	O	O
reference table x Intermediate result	O	O
Intermediate result x Intermediate result	O	O



Distributed Transaction





CITUS에서 분산환경에서 atomic transaction보장을 위해

- Built-in 2pc 이용
- 2pc 취약점을 보안하기 위해 coordinator HA 구성을 추천
- Coordinator는 2PC command을 metadata table에 저장
- 주기적으로 metadata table을 체크하여 문제가 있는 트랜잭션을 복구한다.

2PC 비용은 비싸다

- Reference table의 CUD는 2PC(multi-shard statement)
  - Worker node수가 많을 수록 비용 증가
- Replication 개수가 2이상인 shard의 cud는 2pc(multi-shard statement)
  - Replication 개수가 많아 질수록 비용 증가

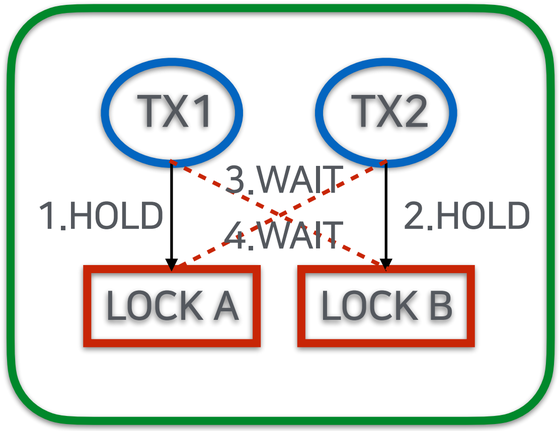
Distributed Transaction

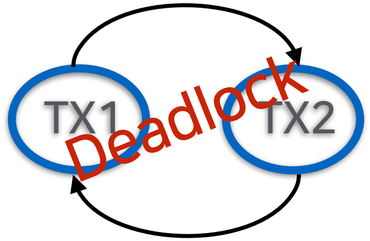
DEVIEW 2020

```
# explain update member set data = 100 where member_id in (20,61);
# explain update member set data = 100 where member_id in (20,60);
( select member_id, 100 as data from member where member_id in (20,60) ) r where p.member_id = r.member_id;
                                                  QUERY PLAN
                                                  QUERY PLAN
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
 Custom Scan (Citus Adaptive)  (cost=0.00..0.00 rows=0 width=0)
 Custom Scan (Citus Adaptive)  (cost=0.00..0.00 rows=0 width=0)
 Custom Scan (Citus Adaptive)  (cost=0.00..0.00 rows=0 width=0)
   Task Count: 32
   Task Count: 2
   Task Count: 1
   Tasks Shown: One of 32
   Tasks Shown: One of 2
   Tasks Shown: All
   ->  Task
   ->  Task
   ->  Task
         Node: host=worker004-shop port=5432 dbname=gshopdb
         Node: host=worker004-shop port=5432 dbname=gshopdb
         Node: host=worker004-shop port=5432 dbname=gshopdb
         ->  Update on member_137948 member  (cost=2.52..4.66 rows=2 width=22)
         ->  Update on member_137948 member  (cost=2.52..4.66 rows=2 width=22)
         ->  Update on member_137948 member  (cost=2.52..4.66 rows=2 width=22)
               ->  Nested Loop  (cost=0.30..9.48 rows=2 width=28)
               ->  Bitmap Heap Scan on member_137948 member  (cost=2.52..4.66 rows=2 width=22)
               ->  Bitmap Heap Scan on member_137948 member  (cost=2.52..4.66 rows=2 width=22)
                     Recheck Cond: (member_id = ANY ('{20,61}'::bigint[]))
                     ->  Index Scan using member_pkey_137948 on member_137948 p  (cost=0.15..4.74 rows=2
width=14)
                     ->  Bitmap Index Scan on member_pkey_137948  (cost=0.00..2.52 rows=2 width=0)
                           Recheck Cond: (member_id = ANY ('{20,60}'::bigint[]))
                           Index Cond: (member_id = ANY ('{20,61}'::bigint[]))
                           Index Cond: (member_id = ANY ('{20,60}'::bigint[]))
                     ->  Bitmap Index Scan on member_pkey_137948  (cost=0.00..2.52 rows=2 width=0)
                           Index Cond: (member_id = ANY ('{20,60}'::bigint[]))
               ->  Index Scan using member_pkey_137948 on member_137948 p  (cost=0.15..2.37 rows=1 width=14)
                     Index Cond: (member_id = member.member_id)
```

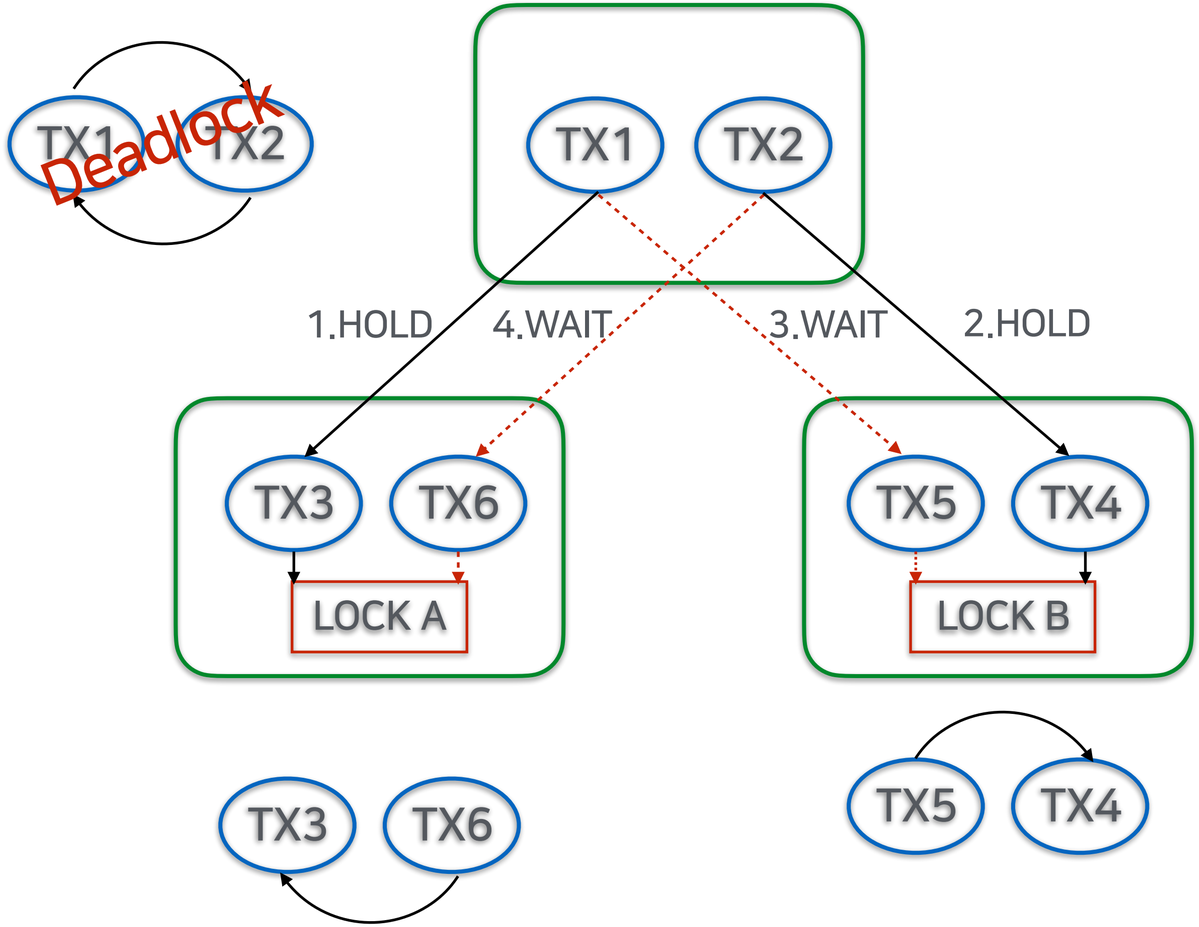
Distributed Deadlock

Deadlock





Distributed Deadlock



Distributed Deadlock

DEVIEW 2020

Detect a distributed deadlock

1. 코디네이터는 주기적으로 모든 워커노드에서 특정시간동안 locks(default 1s)을 기다리고 있는 트랜잭션을 모니터링한다.
2. 만약 해당 트랜잭션이 발견되면 모든 워커노드에서 lock 정보를 가져온다.
3. 해당 lock 정보를 통해 그래프를 만들고, cycle 존재 유무를 판단한다.
4. cycle이 발견되면(deadlock), cycle이 없어지도록 트랜잭션을 종료시킨다.

Prevent a distributed deadlock

Distributed deadlock을 찾는 비용은 매우 비싸니, distributed deadlock을 사전에 미리 방지할 수 있는 방법을 구현

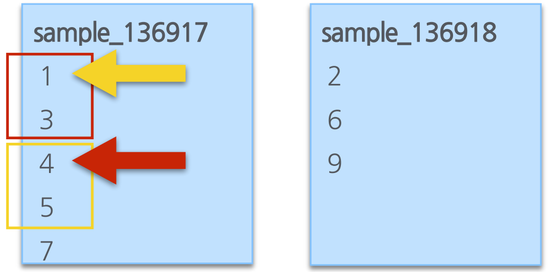
The simplest solution is to only allow one multi-shard transaction at a time.

advisory type의 ShareUpdateExclusiveLock(shard lock)을 생성한다.

Distributed Deadlock

DEVIEW 2020

Example 1. Single shard transaction



Session #1

BEGIN;

UPDATE sample SET data = ‘ses1’ where id in (1,3);

UPDATE2

UPDATE sample set data = ‘ses1’ where id in (4,7);

—- waiting—-

Session #2

BEGIN;

UPDATE sample SET data = ‘ses2’ where id in (4,5);

UPDATE2

UPDATE sample set data = ‘ses2’ where id = 1;

DETAIL: Process 35623 waits for ShareLock on transaction 465702626; blocked by process 51095.

Process 51095 waits for ShareLock on transaction 465703259; blocked by process 35623.

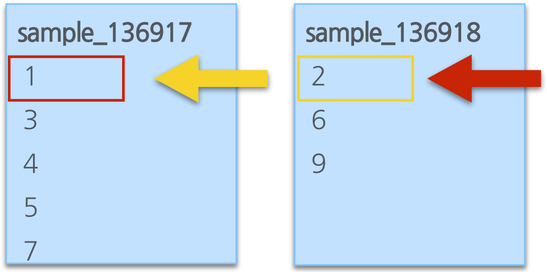
HINT: See server log for query details.

CONTEXT: while executing command on worker-xxxx:5432

while updating tuple (0,39) in relation “sample_136917"

Distributed Deadlock

DEVIEW 2020



Example 2. Multi shard transaction

Session #1

Session #2

BEGIN;
UPDATE sample SET data = ‘ses1’ where id =1;
UPDATE1

BEGIN;
UPDATE sample SET data = ‘ses2’ where id = 2;
UPDATE1

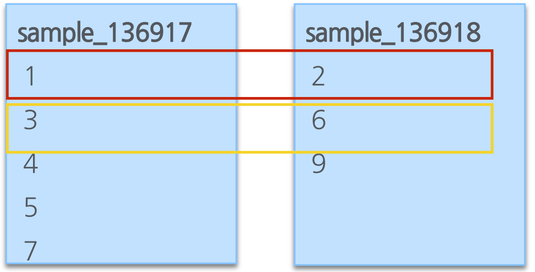
UPDATE sample SET data = ‘ses1’ where id = 2;
—- waiting —-

UPDATE sample SET data=‘ses2’ where id = 1;
ERROR: canceling the transaction since it was involved in a distributed deadlock

Distributed Deadlock

DEVIEW 2020

Example 3. Multi shard transaction



Session #1

BEGIN;

UPDATE sample SET data = ‘ses1’ where id in (1,2);

UPDATE2

```
 locktype | objid  |  pid   |           mode           | granted
----------+--------+--------+--------------------------+---------
 relation |        | 124297 | RowExclusiveLock         | t
 relation |        | 124297 | AccessShareLock          | t
 relation |        | 124297 | AccessShareLock          | t
 relation |        | 124297 | RowExclusiveLock         | t
 virtualxid |      | 124297 | ExclusiveLock            | t
 advisory | 136917 | 124297 | ShareUpdateExclusiveLock | t
 advisory |     10 | 124297 | ShareLock                | t
 advisory | 136918 | 124297 | ShareUpdateExclusiveLock | t
 advisory |     10 | 124297 | ShareLock                | t
(9 rows)
```

Session #2

BEGIN;

UPDATE sample SET data = ‘ses2’ where id in (3,6);

—- waiting —-

Distributed Deadlock

최대한 트랜젝션은 동일 샤드에 영향을 주는 statements만으로 구성하자.

- distributed column 값을 통해 shard를 구분할수 있다
  - hash function만 알수 있다면 가능
  - citus는 오픈소스

```
public static int hashint8(long nvMid) {
    int lohalf = (int)(nvMid);

    final int hihalf = (int)(nvMid >> 32);

    lohalf ^= (nvMid >= 0) ? hihalf : ~hihalf;

    int a, b, c;
    a = b = c = 0x9e3779b9 + 3923099;
    a+= lohalf;

    return calculate(a, b, c);
}

private static int calculate(int a, int b, int c) {
    c ^= b;
    c -= rot(b,14);
    a ^= c;
    a -= rot(c,11);
    b ^= a;
    b -= rot(a,25);
    c ^= b;
    c -= rot(b,16);
    a ^= c;
    a -= rot(c, 4);
    b ^= a;
    b -= rot(a,14);
    c ^= b;
    c -= rot(b,24);
    return c;
}

private static int rot(int x, int k) {
    return (((x)<<(k)) | ((x)>>>(32-(k))));
}
```

key point

- 분산테이블은 동일 distributed column을 사용할수록 유리
- Distributed query planner에 최적화 되도록 쿼리 튜닝 필요
- 성능을 위해 distributed transaction은 만들지 말자
- reference 테이블은 지양
- citus.max_cached_conns_per_worker = 1

  (https://docs.citusdata.com/en/stable/develop/api_guc.html#citus-max-cached-conns-per-worker-integer)

-

CDC(Change Data Capture) 파이프라인

CONTENTS

DEVIEW 2020

1.CITUS 에서 CDC 를 추출하기

1.1 CDC(Change Data Capture)

DEVIEW 2020

CDC

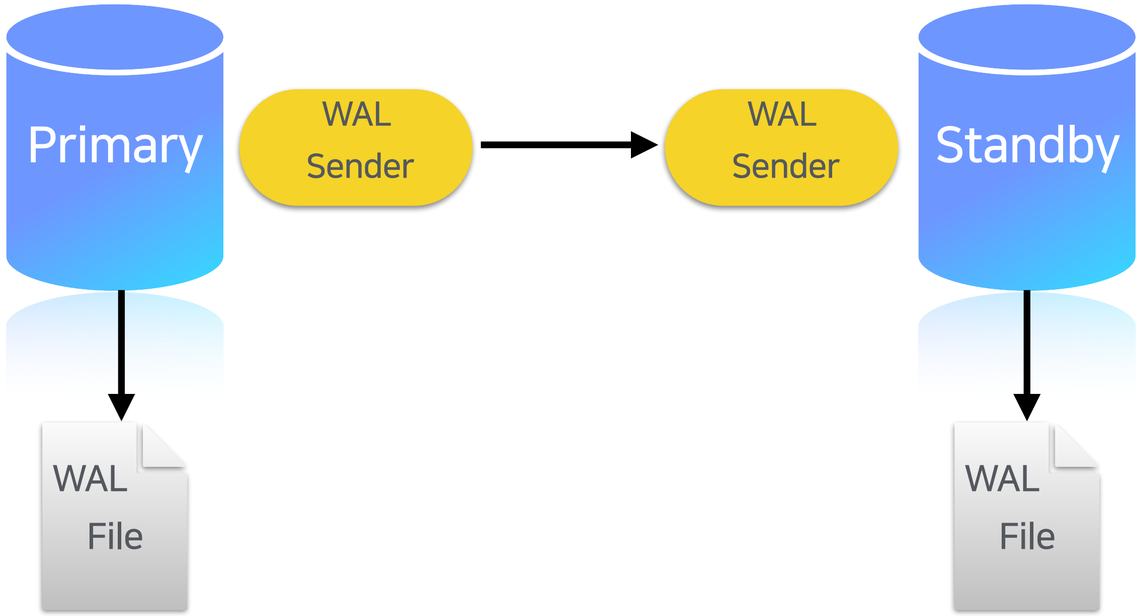
- 데이터베이스의 변경분을 추출하기 위한 프로세스 및 솔루션
- 활용: replication, event driven architecture
- ex) GoldenGate(Oracle), Binlog(MySQL), WAL(PostgreSQL)

Advantages

1. 이종 DBMS 간 복제를 통한 부하 분산
2. Stream 기반 실시간 데이터 파이프라인 처리
3. 데이터 변경 이력 관리

1.2 PostgreSQL 의 Replication

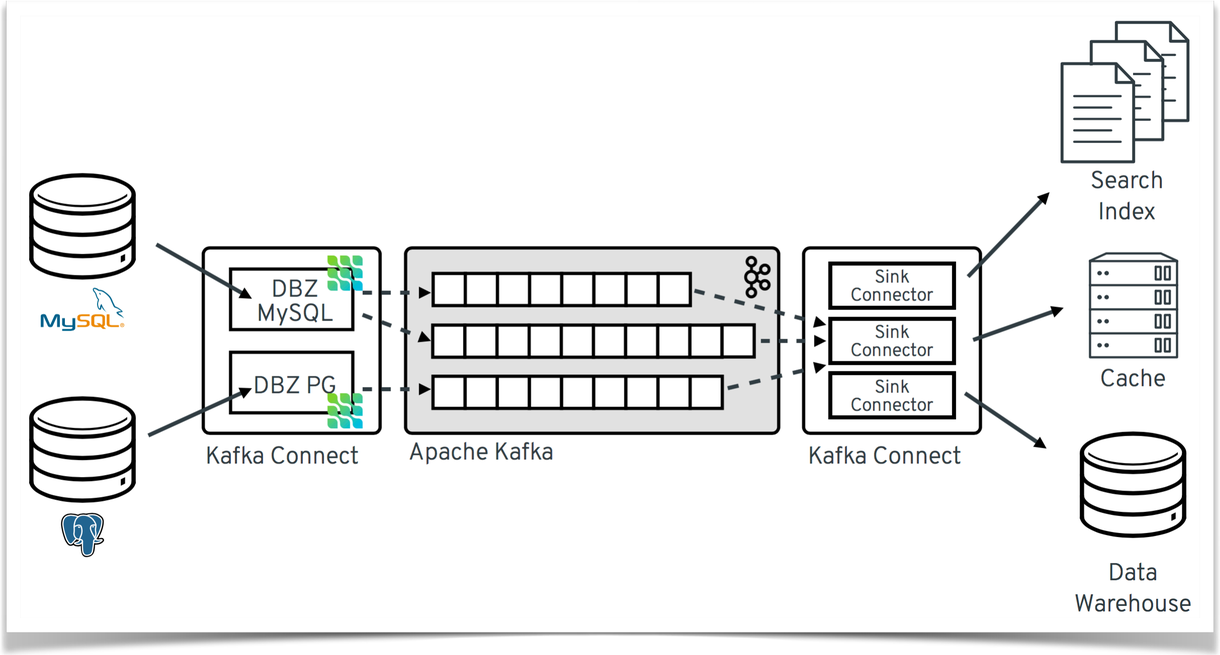
- WAL(Write-Ahead Logging) 를 통해 데이터 복제
- PostgreSQL 9.4 부터 logical replication 추가



1.3 Debezium

DEVIEW 2020

- 다양한 데이터베이스의 CDC 를 추출하기 위한 오픈 소스 플랫폼
- Kafka Source Connector 기반
- 지원하는 데이터베이스
  - MySQL, PostgreSQL, MongoDB 등
- Schema Registry 를 통해 스키마 관리
  - compatibility: FORWARD



reference: https://debezium.io/blog/2019/12/18/debezium-1-0-0-final-released/

1.4 Debezium CDC format

DEVIEW 2020

```
{
 "schema": { ①
   ...
 },
 "payload": { ②
   ...
 },
 "schema": { ③
   ...
 },
 "payload": { ④
   ...
 },
}
```

① key schema

- 테이블의 primary key, pk 가 없을 경우 unique key 에 대한 스키마

② key payload

- 변경 row 의 key 데이터로 key schema 를 따름

③ value schema

- 테이블의 컬럼에 대한 event value 스키마

④ value payload

- 변경 row 의 모든 컬럼에 대한 데이터로 value schema 를 따름

1.4 Debezium CDC format

DEVIEW 2020

value payload

```
{
  "before": null,
  "after": {
    "id": 17,
    "name": "wine glass",
    "price": 17000
  },
  "source": {
   ...
 },
  "op": "c",
  "ts_ms": 1602982061481,
  "transaction": {
    "id": "506",
    "total_order": 9,
    "data_collection_order": 5
  }
}
```

- before: 변경 이전 값
  - 테이블의 REPLICA IDENTITY 를 FULL 로 셋팅 필요
- after: 변경된 값
- op: operation type
  - c: CREATE
  - u: UPDATE
  - d: DELETE
- transaction (provide.transaction.metadata 셋팅 필요)
  - id: string representation of unique transaction identifier
  - total_order: absolute position of the event among all events generated by the transaction
  - data_collection_order: the per-data collection position of the event among all events that were emitted by the transaction

1.4 Debezium CDC format

DEVIEW 2020

Tombstone

- delete 시 value 를 null 로 하여 key event 를 한번 더 보내는 옵션
- kafka log compaction 시 이전 데이터 삭제할 수 있어 효율적
  - ref: https://kafka.apache.org/documentation/#compaction
- tombstones.on.delete: true 설정 (기본값)

key value


```
{"id":2}        {"before":{"cdc._public.product.Value":{"id":2,"name":null,"price":null}},"after":null,"source":{"version":"1.1.0.
Final","connector":"postgresql","name":"cdc","ts_ms":1602993558965,"snapshot":{"string":"false"},"db":"postgres","schema":"public"
,"table":"product_102014","txId":{"long":702},"lsn":{"long":50983992},"xmin":null},"op":"d","ts_ms":{"long":1602993558967},"transa
ction":{"io.confluent.connect.avro.ConnectDefault":{"id":"702","total_order":1,"data_collection_order":1}}}
```



```
{"id":2}        null
```


tombstone

1.5 Debezium Register

DEVIEW 2020

- Debezium connector class 가 포함된 Kafka Source Connector 를 구동
- RESTful API 로 Kafka Source Connector 에 debezium task 를 등록

```
{
  "name": "pg",
  "config": {
    "database.hostname": "__PG_HOST__",
    "database.port": "__PG_PORT__",
    "database.user": "__PG_USER__",
    "database.password": "__PG_PW__",
    "database.dbname": "__PG_DBNAME__",
    "table.whitelist": "__TABLE_WHITE_LIST__",
    "connector.class": "io.debezium.connector.postgresql.PostgresConnector",
    "slot.name": "__SLOT_NAME__",
    "publication.name": "__PUB_NAME__",
    "plugin.name": "pgoutput",
    "snapshot.mode": "never",
    "database.server.name": "cdc",
    "time.precision.mode": "connect",
    "heartbeat.interval.ms": "5000"
  }
}
```

POST /connectors

Debezium Demo

1.6 Debezium for CITUS

DEVIEW 2020

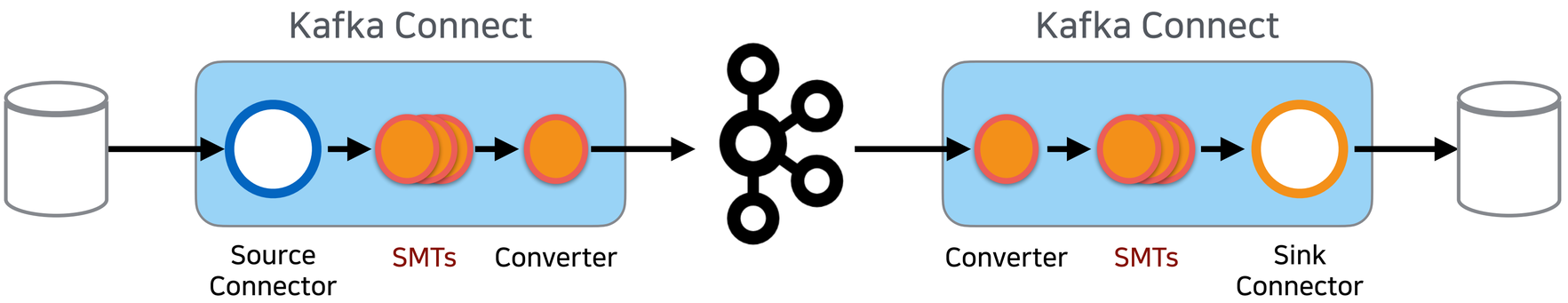
- debezium 은 테이블 단위로 topic 을 생성
- citus 는 multiple shard 기반의 분산 DB 로 테이블이 여러개 존재
- 따라서 테이블 당 shard 개수 만큼의 topic 이 생성되는 문제 발생
- topic 이 많아지고 consumer 관리가 힘들어짐
- table 200 개 x shard 32 개 = 6400 개 topic
- postfix 를 제거하고 하나의 topic 으로 수렴시킬 수 있다면?

1.7 Kafka Connect SMTs

DEVIEW 2020

Single Message Transformations(SMTs)

- kafka connect flow 중간에서 메시지를 변환할 수 있는 모듈
- Cast, Drop 등 기본 제공 모듈도 있고, java 인터페이스 기반으로 직접 제작도 가능



1.7 Kafka Connect SMTs

DEVIEW 2020

RegexRouter

- RegexRouter 를 사용하여 topic 수렴시킬 수 있음
- 추가로 구분자를 dot(.) 에서 underscore(_) 로 변환

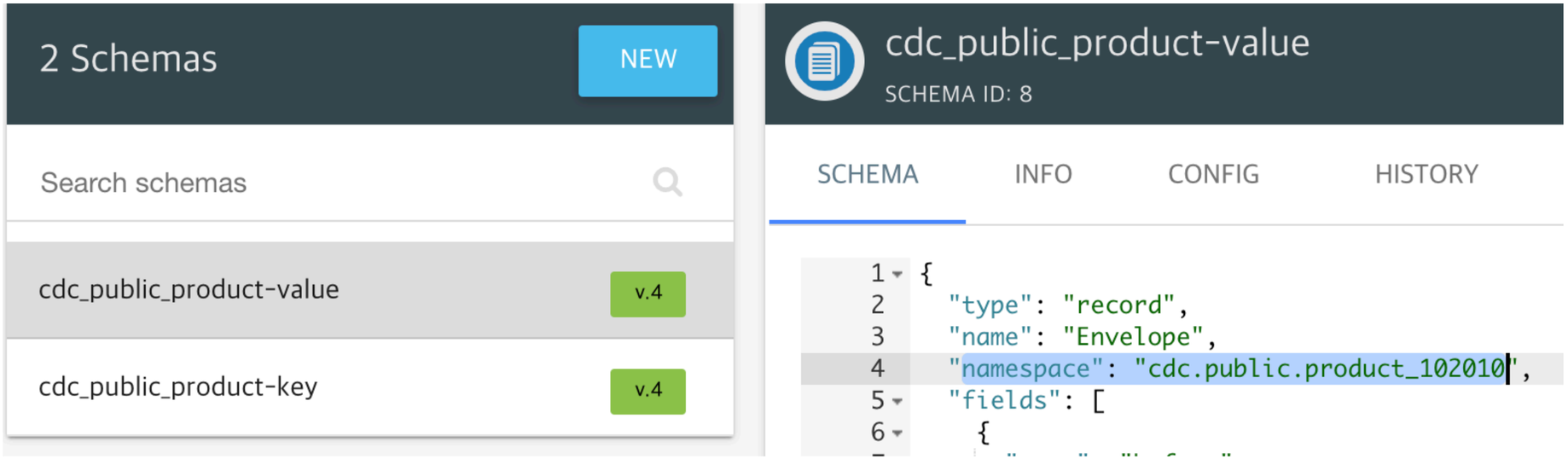
```
  "transforms": "dropPostfix,replace",
  "transforms.dropPostfix.type": "org.apache.kafka.connect.transforms.RegexRouter",
  "transforms.dropPostfix.regex": "(^cdc\\..+\\..+?)(_p\\d+)?_\\d+$",
  "transforms.dropPostfix.replacement": "$1",
  "transforms.replace.type": "org.apache.kafka.connect.transforms.RegexRouter",
  "transforms.replace.regex": "^(cdc)\\.(.+)\\.(.+)$",
  "transforms.replace.replacement": "$1_$2_$3"
}
```

1.7 Kafka Connect SMTs

DEVIEW 2020

Schema Registry Versioning issue

- Schema Registry 버전이 shard 개수 만큼 계속 올라가는 문제
- 스키마의 namespace, connect.name 등의 필드에 shard 이름이 들어있기 때문
- 버전이 메시지 발행수 만큼 무한 증가하지는 않음
  - Schema Registry 가 MD5 Hash 로 동일 스키마인지 구분하여 예전의 ID 를 반환하기 때문
- 그렇다고 해도, 스키마 변경 횟수 x shard 개수 만큼 버전 상승, 관리 어려움

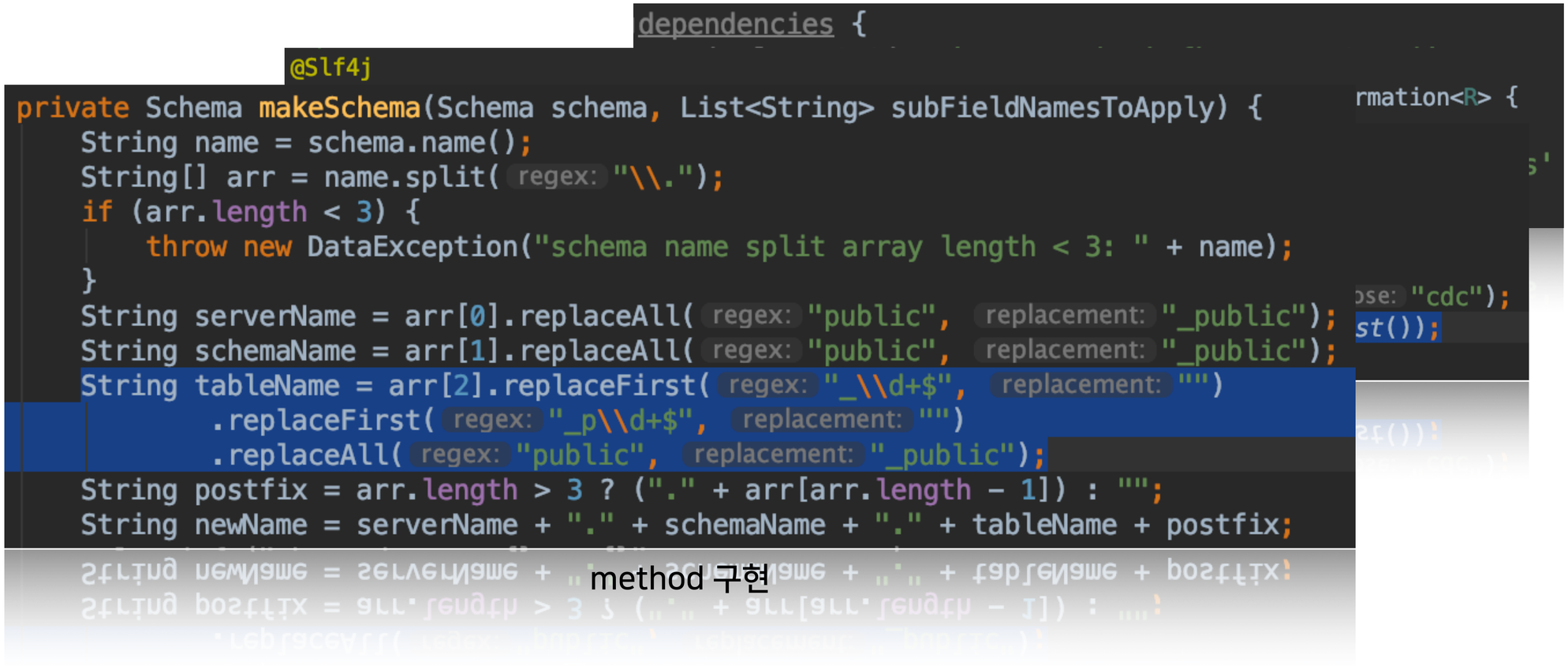


4개 데이터 insert 후 version 이 4까지 올라간 모습

1.8 Apply Custom SMTs

DEVIEW 2020

- Transformation 인터페이스 직접 구현하여 스키마도 변경
- 추가로 topic 명 UPPER CASE 변환 및 public 필드를 _public 으로 치환



1.8 Apply Custom SMTs

DEVIEW 2020

- 빌드된 jar 파일을 debezium 의 libs 경로에 추가
- debezium register API 의 json 에 custom SMTs 적용 (citusCdc)

```
  "transforms": "dropPostfix,replace,citusCdc",
  "transforms.dropPostfix.type": "org.apache.kafka.connect.transforms.RegexRouter",
  "transforms.dropPostfix.regex": "(^cdc\\..+\\..+?)(_p\\d+)?_\\d+$",
  "transforms.dropPostfix.replacement": "$1",
  "transforms.replace.type": "org.apache.kafka.connect.transforms.RegexRouter",
  "transforms.replace.regex": "^(cdc)\\.(.+)\\.(.+)$",
  "transforms.replace.replacement": "$1_$2_$3",
  "transforms.citusCdc.type": "com.naver.csp.data.libs.kafkaconnect.cdc.transforms.CitusCdcConvertor"
}
```

Debezium with Custom SMTs Demo

2.CDC 를 이종 DBMS 에 반영하기

2.1 JDBC Sink Connector

DEVIEW 2020

JDBC Sink Connector

- Kafka 의 cdc 데이터를 DB 에 반영하기 위한 connector
- JDBC driver 를 이용하여 DB access

Data mapping

- Debezium 의 CDC schema 를 Connect Struct 로 변환 필요
- debezium-core 의 transforms 로 제공
  - io.debezium.transforms.ExtractNewRecordState

2.1 JDBC Sink Connector

DEVIEW 2020

Idempotent write

- 멱등성 유지를 위해 upsert 지원
- insert.mode: upsert

Database	Upsert style
MySQL	`INSERT .. ON DUPLICATE KEY UPDATE ..`
Oracle	`MERGE ..`
PostgreSQL	`INSERT .. ON CONFLICT .. DO UPDATE SET ..`
SQLite	`INSERT OR REPLACE ..`
SQL Server	`MERGE ..`
Other	not supported

reference: https://docs.confluent.io/current/connect/kafka-connect-jdbc/sink-connector/index.html

2.3 MySQL Sink Connector Register

DEVIEW 2020

- kafka-connect-jdbc.jar 및 mysql-connector-java.jar 를 kafka-connect lib 경로에 위치
- debezium-core.jar 를 lib 경로에 추가 (for debezium transforms)
- RESTful API 로 Kafka Sink Connector 에 sink task 를 등록

```
{
  "connector.class": "io.confluent.connect.jdbc.JdbcSinkConnector",
  "tasks.max": "1",
  "topics": "CDC_PUBLIC_PRODUCT",
  "connection.url": "jdbc:mysql://dev-bob010-ncl.nfra.io:3306/test",
  "connection.user": "root",
  "connection.password": "qwe123",
  "dialect.name": "MySqlDatabaseDialect",
  "auto.create": "true",
  "pk.mode": "record_key",
  "insert.mode": "upsert",
  "delete.enabled": "true",
  "batch.size": "500",
```

2.3 MySQL Sink Connector Register

DEVIEW 2020

```
    "key.converter": "io.confluent.connect.avro.AvroConverter",
    "key.converter.schema.registry.url": "http://dev-bob011-ncl.nfra.io:18081",
    "value.converter": "io.confluent.connect.avro.AvroConverter",
    "value.converter.schema.registry.url": "http://dev-bob011-ncl.nfra.io:18081",
    "internal.key.converter": "org.apache.kafka.connect.json.JsonConverter",
    "internal.value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "table.name.format": "${topic}",
    "transforms": "dropPrefix,convert",
    "transforms.dropPrefix.replacement": "$1",
    "transforms.dropPrefix.regex": "CDC_PUBLIC_(.*)",
    "transforms.dropPrefix.type": "org.apache.kafka.connect.transforms.RegexRouter",
    "transforms.convert.type": "io.debezium.transforms.ExtractNewRecordState",
    "transforms.convert.drop.tombstones": "false",
    "transforms.convert.delete.handling.mode": "drop"
}
```

MySQL Sink Connector Demo

DEVIEW 2020

3.CDC 파이프라인 모니터링

3.1 JMX Exporter for Prometheus

DEVIEW 2020

Prometheus

- 오픈소스 모니터링/Alert 시스템
- Time-series 기반 다차원 데이터 모델

JMX Exporter

- JMX 으로부터 mBean 을 특정 port 로 expose 하는 java agent
- prometheus 에서 해당 endpoint 로부터 주기적으로 scrape
- kafka-connect 수행 시 JAVA_OPTS 에 다음과 같이 추가하여 사용

```
-javaagent:/usr/share/java/libs/jmx_prometheus_javaagent-0.14.0.jar=8080:config.yaml
```

3.2 Debezium Metric

DEVIEW 2020

MilliSecondsBehindSource

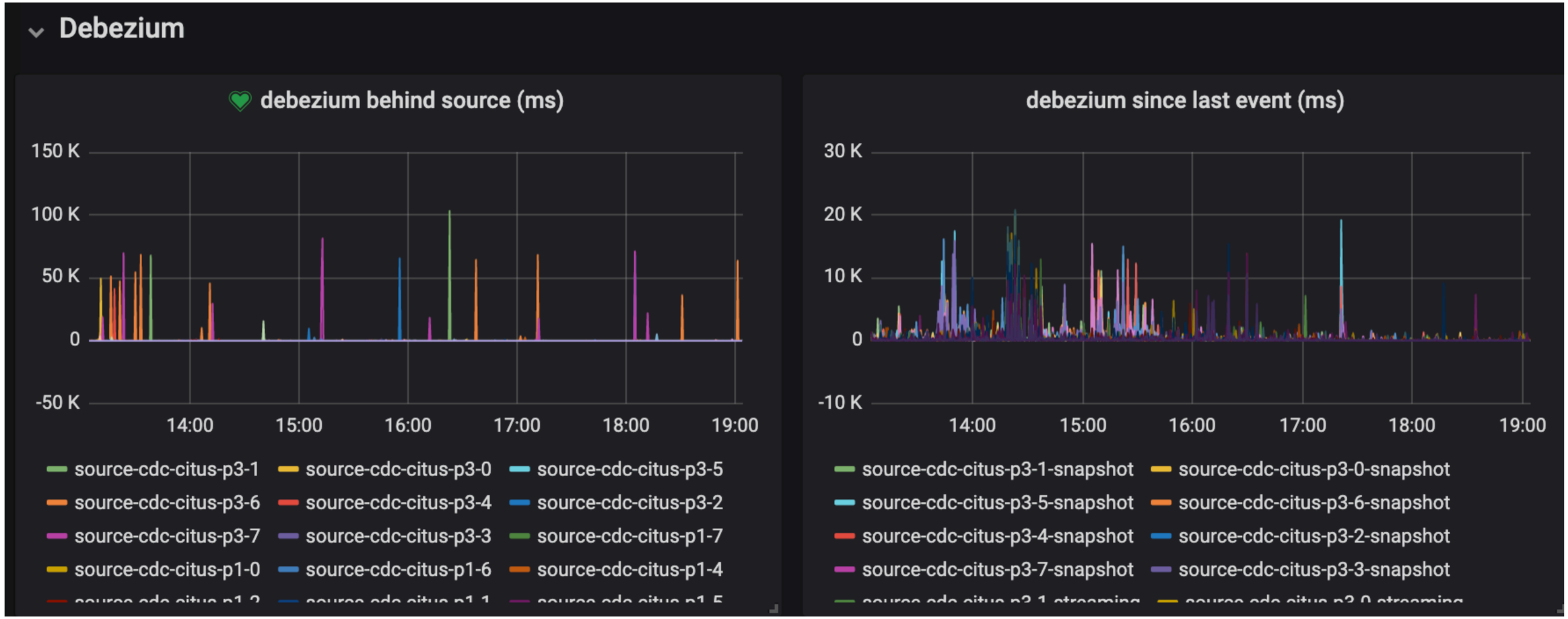
- 마지막 변경 event 의 timestamp 와 debezium 이 읽은 timestamp 의 차이
- 지연이 많아질수록 높은 값

MilliSecondsSinceLastEvent

- 최근 debezium 이 event 를 처리한 시간과 현재 시간의 차이
- DB 변경이 발생하지 않으면 값이 높아짐

3.3 Debezium Monitoring

DEVIEW 2020



3.4 MirrorMaker2 Metric

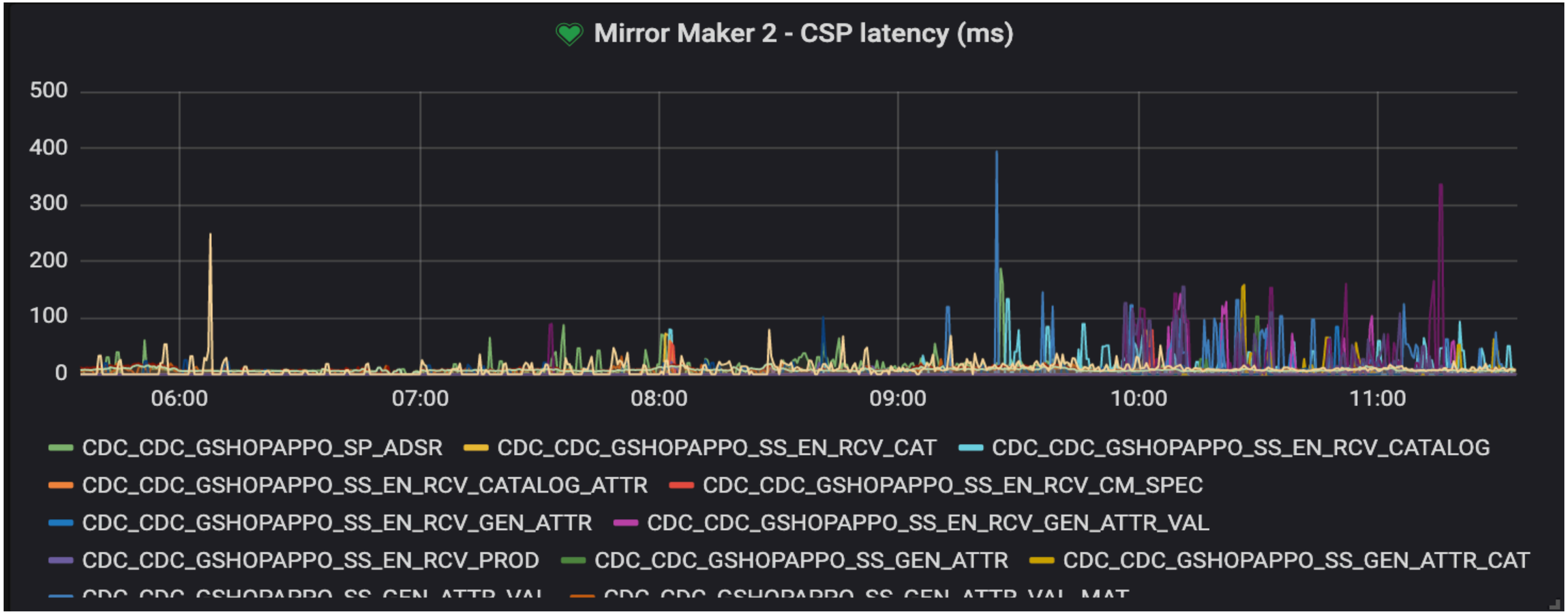
MirrorMaker2

- kafka cluster 를 복제하기 위한 framework
- cdc 용 kafka 를 application 용 kafka 와 격리시키기 위해 사용
- Source Connect 기반으로 별도로 offset 을 기록하여 metric 이 필요

replication-latency-ms-avg

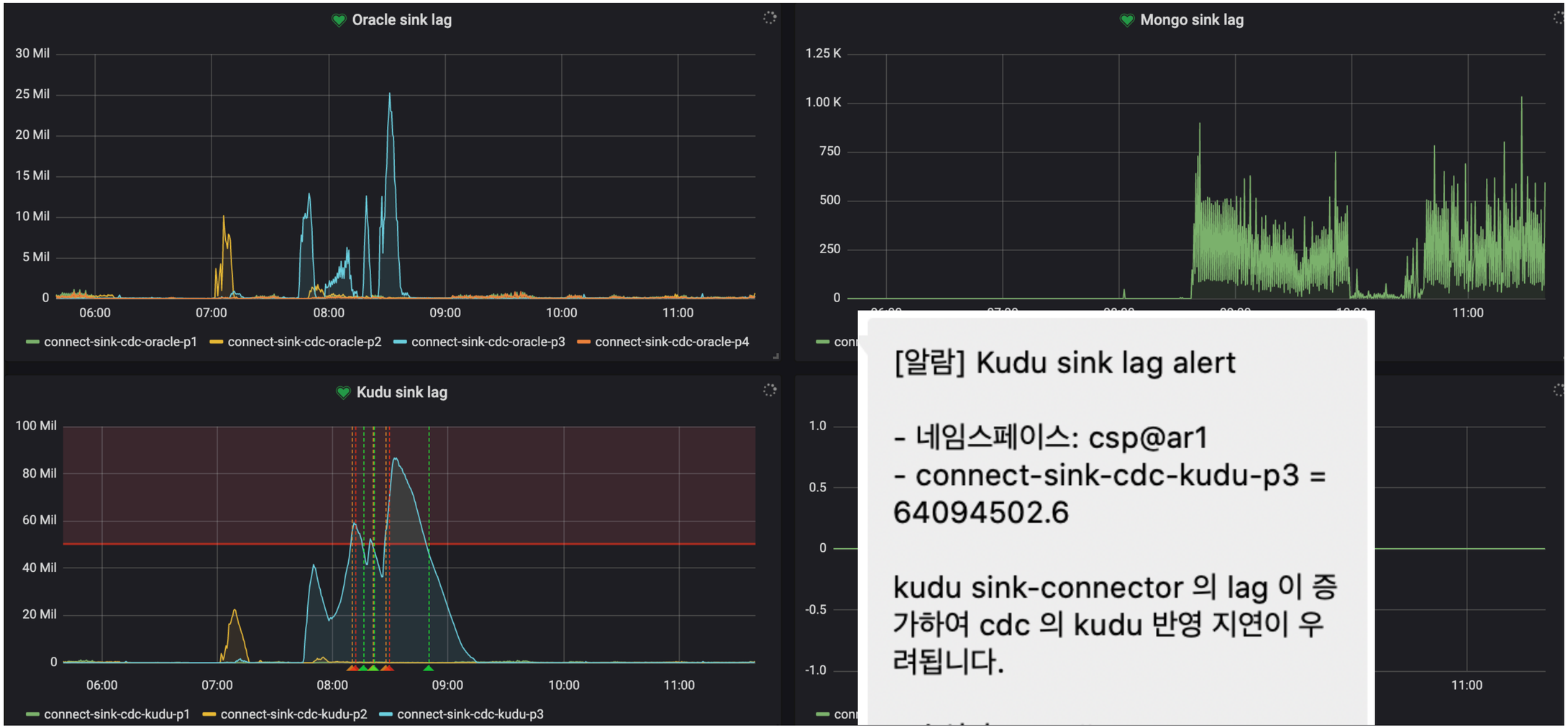
- timespan between each record's timestamp and downstream ACK

3.5 MirrorMaker2 Monitoring



3.6 Sink Connector Monitoring

DEVIEW 2020



Thank You